Chapter 15 Software reuse:

15.1 The reuse landscape

15.2 Application frameworks

15.3 Software product lines

15.4 Application system reuse

Note: Images excluded due to time constraints

15.1 The Reuse Landscape

Overview

- Software **reuse** has evolved over the past 20 years, offering various techniques to support different levels and types of reuse. The choice of reuse technique depends on several key factors, including system requirements, development team expertise, and the software's criticality.

Types of Reuse Techniques

1. **Application Frameworks**: Collections of classes adapted and extended to create systems.
2. **Architectural Patterns**: Standard software architectures used as application bases.
3. **Component-based Software Engineering**: Systems developed by integrating standard components.
4. **Configurable Application Systems**: Domain-specific systems designed for specific customer needs.
5. **Design Patterns**: Generic abstractions represented as patterns.
6. **ERP Systems**: Large-scale systems configured for an organization.
7. **Legacy System Wrapping**: Legacy systems accessed through defined interfaces.
8. **Model-driven Engineering**: Code generated from domain and implementation-independent models.
9. **Program Generators**: Systems generated from user-supplied models.
10. **Program Libraries**: Class and function libraries for common abstractions.
11. **Service-oriented Systems**: Systems developed by linking shared services.
12. **Software Product Lines**: Applications adapted around a common architecture for different customers.
13. **Systems of Systems**: Integration of two or more distributed systems to create a new system.

Key Factors for Planning Reuse

1. **Development Schedule**: For rapid development, reusing complete systems is preferable.
2. **Software Lifetime**: For long-lifetime systems, focus on maintainability and long-term implications.
3. **Team Expertise**: Reuse technologies are complex; focus on areas where the team has expertise.
4. **Software Criticality**: For critical systems requiring certification, having source code access is crucial.
5. **Application Domain**: In domains like manufacturing and medical systems, generic products can often be configured for local needs.
6. **Platform Compatibility**: Some components or systems may be platform-specific.

Generator-based Reuse

- **Generator-based reuse** embeds domain-specific solutions into automated tools, allowing users to create new systems effectively.

Managerial Considerations

- The decision to reuse often hinges more on **managerial willingness** than technical feasibility. Managers may prefer known risks of development over unknown risks of reuse. To promote reuse, it may be necessary to introduce a company-wide reuse program.

15.2 Application Frameworks

Overview

- **Application frameworks** are larger-grain abstractions in object-oriented development that support design reuse and class reuse. They provide a skeleton architecture and reusable components for a family of related applications.

Characteristics of Frameworks

1. **Language-Specific**: Implemented in object-oriented programming languages like Java, C#, C++, Ruby, and Python.
2. **Extensibility**: Can incorporate other frameworks and can be extended using features like inheritance and polymorphism.

Types of Frameworks

1. **Web Application Frameworks (WAFs)**: These are the most widely used and are usually based on the Model-View-Controller (MVC) Composite pattern.
2. **System Infrastructure Frameworks**: Support the development of system infrastructures like communications and user interfaces.
3. **Middleware Integration Frameworks**: Support component communication and information exchange (e.g., Microsoft’s .NET, Enterprise Java Beans).
4. **Enterprise Application Frameworks**: Concerned with specific application domains but have been largely superseded by software product lines.

Components in Web Application Frameworks (WAFs)

1. **Security**: Classes for user authentication and access control.
2. **Dynamic Web Pages**: Classes for defining web page templates.
3. **Database Integration**: Abstract interface to different databases.
4. **Session Management**: Classes for creating and managing user sessions.
5. **User Interaction**: Support for AJAX and/or HTML5, device-independent interfaces.

Implementation Considerations

- **Inversion of Control**: Framework objects control the flow, invoking “hook methods” in response to events, which are then linked to user-provided functionality.
- **Callbacks**: Methods that are called in response to events recognized by the framework.

Challenges and Limitations

- **Complexity**: Frameworks are inherently complex and require time to learn.
- **Evaluation**: Difficult and expensive to evaluate and choose the most appropriate framework.
- **Debugging**: More challenging due to the complexity and interactions between framework methods.

15.3 Software Product Lines

Overview

- **Software Product Lines** are sets of applications with a common architecture and shared components, adapted for specific customer requirements. They are effective for supporting multiple similar systems, thus enabling high levels of code and test reuse.

Core Elements of a Product Line

1. **Core Components**: Infrastructure support that usually remains unchanged.
2. **Configurable Components**: Can be modified and configured for new applications.
3. **Specialized Components**: Domain-specific elements that may be replaced in new instances.

Development Process

1. **Elicit Stakeholder Requirements**: Use an existing system to gather requirements.
2. **Select Closest-Fit System**: Choose an existing product closest to the new requirements.
3. **Renegotiate Requirements**: Make adjustments based on planning and new details.
4. **Adapt Existing System**: Develop new modules and adapt existing ones.
5. **Deliver New Product**: Document its features for future reuse and perform any deployment-time configurations.

Types of Specialization

1. **Platform Specialization**: Adapt for different operating systems.
2. **Environment Specialization**: Adapt for different hardware and peripherals.
3. **Functional Specialization**: Adapt for specific customer needs.

4. **Process Specialization**: Adapt to specific business processes.

Configuration Stages

1. **Design-Time Configuration**: Modification of the core product line for creating new systems.
2. **Deployment-Time Configuration**: A generic system is specialized at the time of deployment using configuration data.

Comparison with Application Frameworks

1. **Object-Oriented Features**: Frameworks rely on inheritance and polymorphism; product lines may not.
2. **Domain-Specificity**: Frameworks are usually general; product lines are often domain-specific.
3. **Hardware Support**: Product lines often support hardware interfacing; frameworks usually don't.
4. **Ownership**: Product lines are usually owned by the same organization and adapted from the closest family member.

Challenges and Considerations

- **Balance between Reuse and Customization**: Striking a balance can lead to quicker delivery and lower costs.
- **Reconfiguration Complexity**: Deployment-time configuration can be complex and time-consuming.

Tools and Support

- Software tools like configuration planning tools may support the complex process of deployment-time configuration.

15.4 Application System Reuse

Overview

- **Application System Products** are software systems adapted for multiple customers without changing the source code. Also known as COTS (Commercial Off-the-Shelf System) products, they are designed for general use and thus have high reuse potential.

Advantages of Application System Reuse

1. **Rapid Deployment**: Faster rollout of reliable systems.
2. **Transparency**: Easier to judge suitability due to existing implementations.
3. **Reduced Risks**: Mitigates some development risks.
4. **Resource Efficiency**: Businesses can focus on core activities.
5. **Technology Updates**: Simplified, as they are the vendor's responsibility.

Challenges and Risks

1. **Adapting Requirements**: May necessitate changes to existing business processes.
2. **System Assumptions**: Businesses might need to adapt to the system's underlying assumptions.

3. **System Selection**: Difficult due to poor documentation, wrong choices can be costly.
4. **Lack of Local Expertise**: Dependence on vendor or external consultants.
5. **Vendor Control**: Risks related to vendor business stability or changes.

Types of Application Systems

1. **Individual Systems**: Single-vendor generic applications tailored to customer needs.
2. **Integrated Systems**: Combination of functionalities from multiple vendors to create a new system.

Focus Areas in Development

- **Individual Systems**: Focus is on **system configuration**.
- **Integrated Systems**: Focus is on **system integration**.

Maintenance Responsibility

- **Individual Systems**: Vendor is responsible for maintenance.
- **Integrated Systems**: System owner is responsible for maintenance.

Platform Provision

- **Individual Systems**: Vendor provides the platform.
- **Integrated Systems**: System owner provides the platform.

Summary of Differences

- **Individual Systems** are based on a generic solution and standardized processes, while **Integrated Systems** offer flexible solutions for customer processes.

Application System Integration

- Further discussion on integration is mentioned to be covered in Section 15.4.2.

15.4.1 Configurable Application Systems

Overview

- **Configurable application systems** are generic software designed to support specific business types, activities, or even entire enterprises.
- These systems include built-in assumptions about how users work, which can sometimes lead to issues.
- **Enterprise Resource Planning (ERP) systems** are a specific example, designed to integrate various business functions like inventory management, manufacturing, and customer relationship management.

Types of Configurable Systems

1. **Domain-Specific Systems**: Target a particular business function, like document management.
2. **ERP Systems**: Large-scale systems that can support multiple business functions.

Key Features of ERP Systems

1. **Modules**: ERP systems have modules to support different business functions such as purchasing, supply chain management, logistics, and customer relationship management (CRM).
2. **Business Process Models**: Each module has associated business process models to define roles and activities.
3. **Common Database**: A single database maintains information for all business functions, eliminating data replication.
4. **Business Rules**: A set of rules that apply to all data to maintain consistency across business functions.

Configuration Process

1. **Select Modules**: Choose the necessary modules based on business needs.
2. **Data Model**: Define how the organization's data will be structured in the system database.
3. **Business Rules**: Establish rules that apply to data.
4. **External Interactions**: Define how the system will interact with external systems.
5. **Input/Output Design**: Design the input forms and output reports.
6. **New Business Processes**: Adapt or design business processes to fit the system's underlying model.
7. **Deployment Settings**: Set parameters for system deployment.

Limitations and Challenges

- **Functionality**: Limited to the built-in modules of the ERP system.
- **Business Model Mismatch**: The configuration language may not align with the customer's business model, causing potential issues.
- **Complex Configuration**: Requires detailed knowledge and often the help of consultants.

Testing Challenges

1. **Test Automation**: May be difficult or impossible due to lack of access to an API.
2. **Subtle Errors**: Errors often arise from misunderstandings between those configuring the system and the end-users, and are specific to business processes.

15.4.2 Integrated Application Systems

Overview

- **Integrated application systems** combine two or more application systems or legacy systems to meet specific needs.
- These systems may interact via APIs, service interfaces, or through data exchange mechanisms.

Design Choices for Integration

1. **Selecting Appropriate Systems**: Choose the most suitable application systems that offer the required functionality.
2. **Data Exchange**: Handle different data structures and formats by writing adaptors for conversion.
3. **Feature Utilization**: Decide which features of each application system are needed and disable unused functionalities to avoid interference.

Case Study: Procurement System

- Illustrates how a large organization saved time and costs by integrating a legacy ordering system with a web-based e-commerce platform and an email system.
- Utilized adaptors to handle data format conversions and notifications.

Service-Oriented Approach

- **Service Wrappers**: Use a standard service interface for each unit of functionality. These wrappers can particularly benefit legacy systems during integration.

Challenges and Problems in System Integration

1. **Lack of Control over Functionality and Performance**: System may not be implemented properly or may have hidden operations.
2. **System Interoperability Issues**: Systems may have their own assumptions making integration difficult.
3. **No Control over System Evolution**: Vendors frequently update their systems, which might not be compatible with previous versions.
4. **Vendor Support Variability**: Level of support can differ among vendors and may change over time due to various factors like market pressures.

Life-Cycle Problems

- Challenges in integration do not only affect initial development but can extend into the maintenance phase, especially when original developers are no longer involved.

Key Points

- Software reuse through application system integration can significantly reduce costs and development time.
- However, it comes with challenges like lack of control over functionality, system interoperability issues, and variable vendor support.

Summary

- There are many different ways to reuse software. These range from the reuse of classes and methods in libraries to the reuse of complete application systems.
- The advantages of software reuse are lower costs, faster software development, and lower risks.
- System dependability is increased. Specialists can be used more effectively by concentrating their expertise on the design of reusable components.
- Application frameworks are collections of concrete and abstract objects that are designed for reuse through specialization and the addition of new objects. They usually incorporate good design practice through design patterns.
- Software product lines are related applications that are developed from one or more base applications. A generic system is adapted and specialized to meet specific requirements for functionality, target platform, or operational configuration.

- Application system reuse is concerned with the reuse of large-scale, off-the-shelf systems.
- These provide a lot of functionality, and their reuse can radically reduce costs and development time. Systems may be developed by configuring a single, generic application system or by integrating two or more application systems.
- Potential problems with application system reuse include lack of control over functionality, performance, and system evolution; the need for support from external vendors; and difficulties in ensuring that systems can interoperate.